Enhancing the Flexibility of Workflow Execution by Activity Anticipation

D. Grigori[1], F. Charoy[2], C. Godart[2]

[1]Université Versailles St-Quentin & INRIA, 45, avenue des Etats-Unis, 78035 Versailles Cedex

daniela.grigori@prism.uvsq.fr

[2]Université Henri Poincaré & INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre Cedex, France

{francois.charoy, claude.godart}@loria.fr

Abstract. This paper introduces an evolution to classical workflow model that allows more flexible execution of processes while retaining its simplicity. On the one hand it allows to describe processes in the same way that they are in design and engineering manuals. On the other hand it allows to control these processes in a way that is close to the way they are actually enacted. This evolution is based on the concept of anticipation, i.e. the weakening of strict sequential execution of activity sequences in workflows by allowing intermediate results to be used as preliminary input into succeeding activities. The concept can also be applied to provide rapid feedback to preceding activities, and for complex sub-processes. The architecture and implementation of a workflow execution engine prototype allowing anticipation is described.

Keywords: workflow flexibility, cooperation, co-design and co-engineering processes, process modeling

1. Introduction

Current workflow models and systems are mainly concerned with the automation of administrative and production business processes. These processes coordinate well-defined activities that execute in isolation, i.e. synchronize only at their *start* and *terminate* states. If current workflow models and current workflow systems apply efficiently for this class of applications, they show their limits when one wants to model the subtlety of cooperative interactions as they occur in interactive or creative processes, typically co-design and co-engineering processes. Several research directions are investigated to provide environments that are more adaptable to user

habits. These directions are described in section 2. They propose most of the time complex evolutions of the basic workflow model. These evolutions may be hard to understand and to master by end users. Thus, our approach consists in adding flexibility to workflow execution with minimal changes of the workflow model. We try to reach this goal by relaxing the way the model is interpreted; users can take some initiative regarding the way they start the assigned activities, leaving the burden of consistency management to the execution engine.

In this paper, we introduce the idea of *anticipation* as a way to support more flexible execution of workflows. The principle is to allow an activity to start its execution even if all "ideal" conditions for its execution are not fulfilled. Anticipation is very common in creative applications: reading a draft, starting to code without complete design, illustrate this idea. Anticipation allows to add flexibility to workflow execution in a way that can not be modeled in advance. It can also be used to accelerate process execution as it increases parallelism in activity execution.

This work is part of the COO project which aims at providing a flexible support for cooperative processes in co-engineering applications. We developed a cooperative workflow model and implemented a prototype that was used in different projects. This paper presents the flexibility of the control flow (the first results were presented in [12]). The integration of this workflow model with a transaction manager in order to allow flexible data exchanges is presented in [10].

The paper is organized as follows. In the next section we motivate our approach providing requirements for a cooperative workflow. In section 3 we present an overview of existing approaches for workflow flexibility. In section 4 we describe our view of anticipation in workflows and the constraints related to it in order to ensure consistent execution of a process. Section 5 is dedicated to the implementation of a workflow engine allowing anticipation. Finally, section 6 concludes.

2. Requirements for co-design and co-engineering applications.

Traditional workflow management systems impose an end-start dependency between activities [28]. This means that an activity can be started only after the preceding ones have completed. If this dependency is well suited for modeling production and administration processes, it is too restrictive for cooperative applications like co-design and co-engineering. This section introduces some requirements to support the flexibility necessary for the creative applications we consider. Then it enhances two upper level design criteria on which our approach is based.

Need for anticipation Most of the time, in a cooperative process activities overlap and start their work with some intermediate[1] results (in opposition to final results[2]) of

[1] Intermediate result: result produced by an activity during its execution, before its end.
[2] Final result: result produced by an activity when it completes.

preceding activities, even if all conditions for their execution are not completely fulfilled :

- preceding activities are not terminated (work on draft or partial results)
- activation conditions are currently evaluated as false or cannot be checked, (some visa is missing, some tests have not been passed)
- input data is not complete (but available data are sufficient to start working)

Allowing activities to start their execution in such a condition is what we call *anticipation*. Figure 1 depicts an execution of the same process without (1) and with anticipation (2). In the second case, the Edit activity provides an intermediate draft of the edited document to Review: Review and Modify activities can be started earlier. Thus the whole process will probably terminate earlier.

This type of execution has the goal to accelerate the process execution and to provide rapid feedback. The example of Figure 1 illustrates how the "Review" activity can provide early comments while the "Edit" activity is still running.

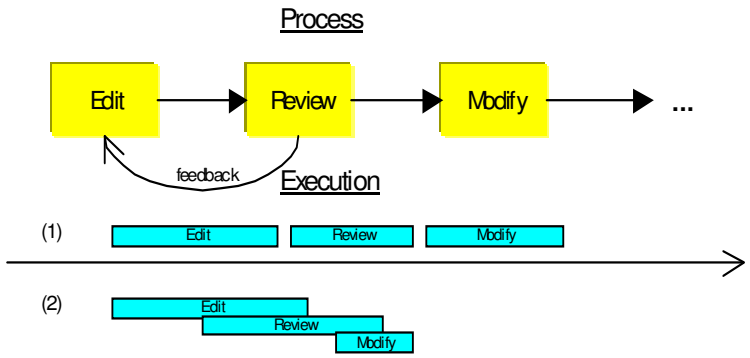

Figure 1 Execution without (1) and with (2) anticipation.

Need to increase parallelism between sub-contractors. We illustrated how anticipation allows process execution to be accelerated. This is especially the case when an activity breaks down into one complex sub-process, the activities of which being executed by different sub-contractors acting on different sub-sets of the input data-set. Figure 2 illustrates this idea : the review of a composite document is on the responsibility of several actors depending on the document part and the actor role: in some cases, they can start their work as soon as they get their corresponding part, and work in parallel.

We can notice that this applies also for production applications as Figure 3, taken from [15] illustrates. The figure represents a typical process to handle the delivery of products by a retail company. The G*et item* activity takes care of the stock control and it is implemented as a sub-process which verifies if the item is available and

otherwise orders it; finally an invoice is produced. The overall process, shown in the upper part of the figure, starts with the initial placement of the order, followed by the get item task explained above. After the products have been extracted from the warehouse, an acknowledgment is sent to the customer and the product is shipped. As the process is considered as a black box, the top level process waits the complete subprocess to finish before to proceed. This decreases the parallelism and increases the reponse time of the system. For instance, the acknowledgment to the customer could be sent as soon as it is known that the product is on stock. Similar, it is not necessary that the shipping activity wait until the whole warehouse administration process concludes.

In [15] the parallelism between the top level process and its subprocess is increased by using a supplementary type of control flow connector, an event based control connector. This connector reacts to the occurrence of events and can be combined with the classical connector, which reacts only to the event corresponding to completion of activities. In section 3 we explain how anticipation allows to increase the parallelism without changing the process model.

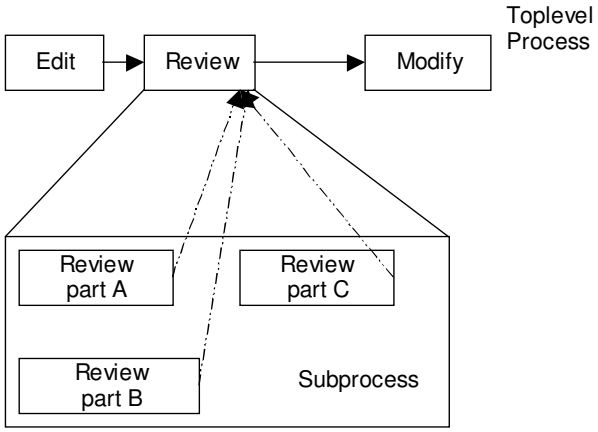

Figure 2 Parallelism between sub-contractors (example 1).

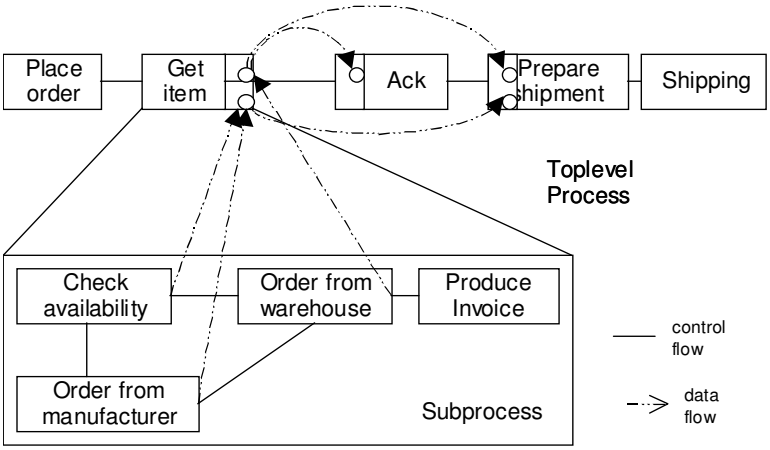

Figure 3 Parallelism between sub-contractors (example 2)

Need to allow intermediate results exchange

"Intermediary objects produced during design activity are significant because they allow the coordination between designers and because they must be considered as real cognitive actors of design" [9]. In other words, intermediate results are first order artifacts for cooperation support.

As introduced above, the main principle on which our vision is built is that people cooperate by exchanging drafts, sketches, intermediate results in order to increase efficiency of the virtual team work : anticipation of work, rapid feedback, better cohesion. Of course, intermediate results are progressively elaborated and a cooperation object will exist in several versions before the final value will be produced. To preserve enough privacy, the initiative to make an intermediate result visible is on the responsibility of human agents. This model is based on some usage analysis, including our proper experience in software development, and discussions with designers in the domain of house building and car manufacturing.

Need for a simple workflow model

Cooperative processes are knowledge intensive processes and they are mainly composed by activities executed by humans. The process model is the basic support for the awareness of process participants. For this reason, the model must be simple, and its interpretation straightforward. Cooperative processes are unstructured and

need dynamic modifications. A simple model facilitates user intervention for dynamic modifications.
Moreover, a well defined workflow system should manage as well structured processes as cooperative processes. Several researchers [21],[4] pointed out that workflow systems should be made more flexible and the goal of leading edge systems is to have variable coordination support being able to support structured and unstructured workflow (or some customized combination of both on a case-by-case basis).

2.2. Design criteria for co-design and co-engineering workflow.

Our response to these requirements is, on the one hand to develop an execution model including the anticipation idea to provide more flexible execution, on the other to render the exchange of data between activities more flexible.

Flexible execution. Parallelism of execution between interacting activities is not really considered in traditional workflow systems and workflow engines. One way to surpass this limit is to change the workflow model, either by incorporating new synchronization modes in the model, or by allowing the model to change dynamically, or both. Another way is to conserve traditional workflow model (WfMC [28] model can be taken as a good reference), but to change the workflow engine, by interpreting workflow scripts in a flexible way. *Based on the consideration that production, administrative and co-design, co-engineering high level process descriptions are done in the same way, with similar graphical representation, we think that the best approach is to conserve traditional workflow models, but to change the workflow engine to increase parallelism and interactions between activities.*

Flexible data management.
While in traditional workflow models activities execute as black boxes that consume data input and produce data output, above requirements request more dynamic interactions between activities. This includes visibility of intermediate results: an activity must be allowed to make visible its results while executing, i.e. before to terminate.

3 Related work

Workflow flexibility has been studied extensively as a mean to improve workflow management systems acceptance [19]. Based on the role and the use of process model we distinguish three main approaches for flexibility.
The first approach considers *the process as a resource for action* [25]. Basically, it means that the process is a guide for users upon which they can build their own plan. It is not a definitive constraint that has to be enforced. Thus, users keep the initiative to execute their activities. They are not constrained by the predefined order of activities but are inspired by it and encouraged to follow it.

Authors of [21] propose to enhance the workflow model with goal activities and regions in order to allow its use as a resource for action. A goal node represents a part of the procedure with an unstructured work specification; its description contains goals, intent or guidelines. Authors of [1] argues that a plan as a resource for action must support users awareness, helping them to situate in the context of the process, either to execute it or to escape to it in order to solve a breakdown.

The second approach uses the process as a constraint for the flow of work, but it is admitted that it may change during its lifetime. The process can be dynamically adapted during its execution. ADEPTflex [22], Chautauqua [5], WASA [27] and WIDE [3] provide explicit primitives to dynamically change running workflow instances. These primitives allow to add/delete tasks and to change control and data flow within a running workflow instance. Constraints are imposed on the modifications in order to guarantee the syntactic correctness of the resulting process instance. [6] [23] [26] propose methods to migrate instances from the old process definition to the new one without introducing errors.

The third approach consists in evolving the process model itself to allow for more flexible execution. In this case, flexibility has to be modelled and is anticipated during the process modelling step. This is one of the branches that are followed by the COO project [8] and by other similar work [7], [24]. In Mobile [16], the authors define several perspectives (functional, behavioral, informational, organizational, operational) for a workflow model, the definitions of perspectives being independent of one another. Descriptive modeling is defined as the possibility to omit irrelevant aspects in the definition of a perspective. In [7], [8], [24] other examples of descriptive modeling are presented as techniques for compact modeling. The authors propose simple modeling constructs that better represent real and complex work patterns to be used, instead of a composition of elementary constructs.

The first two approaches consider flexibility at the level of the process execution itself. In one case, the model is a guide to reach a goal; in the other case, the model is a path to reach a goal that may change during its course. In the third approach, it is the model that evolves to provide the requested flexibility.

In this paper, we consider a fourth way which is not based on the way the process model is used or instantiated, neither on the way it can be evolved or modelled, but which adds flexibility in the workflow management system execution engine itself. This has the advantage of retaining the simplicity of the classical model and may also be adapted to other approaches. Our point of view is that the models that have been already proposed are most of the time very complex and not easy to grasp for designer and end user. Our proposal is a first step toward a simple model that could support a more flexible execution suited to engineering processes.

The exchange of intermediate results between activities is supported in some workflow models [15], [18].

In [15], an event-based communication mechanism allows the exchange of information between activities or sub-processes. Thus, activities in a sub-process that depend on the results of activities in another sub-process, can be started as soon as appropriate messages are received, and do not have to wait until the whole sub-process is finished. While, here, the events are used to start an activity, we allow the exchange of information all the time during activities execution.

[18] proposes a behavior definition for a task as the basis for modeling less-restrictive workflows as well as supporting dynamic workflow changes. New control flow dependencies can be specified; in particular the simultaneous dependency allows controlling data exchange between simultaneous active tasks, assuring that the dependent task does not terminate before the preceding task. In contrast with our approach, the exchange of intermediate results is included in the model, though anticipated during process modeling step.

4 Flexibility of the execution model

In this part we describe the evolution of the workflow execution engine to support flexible process execution and data exchanges, and how we tackle the consistency problems that arise. In the following, the basic workflow model is presented.

4.1 Basic model

The workflow model that we use is very simple, providing the basics for control and data flow modeling. We provide here a brief description that will allow us to explain the evolution we propose on the workflow engine.

The workflow model is based on process graphs. A process is represented as a directed graph, whose nodes are activities. Edges have associated transition conditions expressing the control flow dependencies between activities. Activities are units of work. Each activity has an associated staff query that specifies the resources that can execute it.

An activity having more than one incoming edge is a join activity; it has an associated join condition. For an *or-join* activity the associated join condition is the disjunction of conditions associated to incoming edges. For an *and-join* activity the associated join condition is the conjunction of conditions associated to incoming edges. An activity having more than one outgoing edge is a fork activity.

Activities have input data elements and produce output data elements. The circulation of data between activities is represented by edges between output elements of an activity and input data element of another activity. The consumer must be a direct or transitive successor of the producer activity.

In summary, a process model is represented by a directed graph whose nodes are activities and whose edges represent control flow and data flow constraints.

4.2 Anticipation

As introduced above, traditional workflow management systems impose an end-start dependency between activities [28]. This means that an activity can be started only after the preceding ones have completed. However, in cooperative processes most of the time, successive activities overlap; activities start their work even if preceding activities are not yet terminated.

Anticipation is the mean we propose to support this natural way to execute activities while retaining the advantage of explicit coordination. Anticipation allows an activity to start its execution earlier regarding the control flow defined in the process model. When preceding activities are completed and all activation conditions are met, an anticipating activity enters the normal executing state, i.e. it continues its execution as if it never anticipated. At this time, final values of its input parameters are available. Having already been started, the activity is able to finish its execution earlier. The anticipation allows a more flexible execution, preserving, at the same time, the termination order of activities.

Example of Figure 1(2) is an execution with anticipation. The *Edit* activity provides an intermediate draft of the edited document to *Review*. In this case *Review* and *Modify* activities can be started earlier. The whole process can thus be terminated earlier.

4.3 Execution states of an activity

The possibility to anticipate requires some modifications of the workflow execution model. Our approach is to extend the traditional model (we start from the model defined in [20]) to take into account anticipation. Two new activity states are added: *ready to anticipate* and *anticipating*. The *ready to anticipate* state indicates that the activity can start to anticipate. When an agent having the adequate role chooses it from its *to do list*, the activity enters the *anticipating state*. Figure 4 depicts a state transition diagram including the new added states. Note that before to complete, even if an activity has started to anticipate, it must pass through executing state.

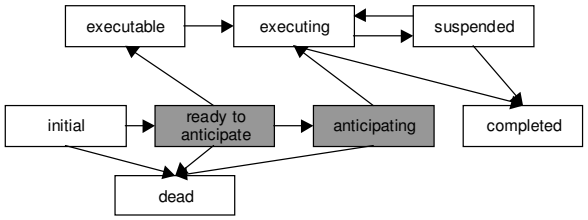

Figure 4 State transition diagram for activities

When a process is started, all activities are in *initial* state, except the start activities (activities with no incoming edges), that are in *executable* state.

Transition from *initial* to *ready to anticipate* state. Concerning the moment when an activity in initial state can start to anticipate, several strategies can be considered:

1. *Free anticipation* – an activity in *initial* state may anticipate at any moment (*ready to anticipate* state merged with *initial* state). This allows activities to start their work earlier. In other words, control flow dependencies defined in the process model are interpreted at execution time as end-end dependencies; i.e. an activity

can finish its execution only when the preceding one has. In our example, it would mean that *Modify* could start at any time. Free anticipation should be reserved to very special cases.

2. *Control flow dictated anticipation* – an activity may anticipate when its direct predecessor has started to work, i.e. is in *anticipating* or *executing* state. For an *or-join* activity[3], at least one of its predecessors must be in the *anticipating* or *executing* state. For an *and-join* activity[4], all the preceding activities must have been started. In this case, the traditional start-end dependency between activities is relaxed, being replaced with a start-start dependency; i.e. the successor activity can start its execution as soon as the predecessor has started. In our example, that means that *Modify* can be started as soon as *Review* has been started. As we explained before, the end-end dependency is ensured for all the anticipation strategies.
3. *Control flow and data flow dictated anticipation* – an activity can anticipate when its predecessors are in *anticipating*, *executing* or *completed* state and for all its mandatory inputs, there are values available. This requirement ensures that the anticipating activity has all its input elements available and its predecessors have already started to work. This is like control flow anticipation with additional constraints concerning inputs existence. In this case, the *Modify* activity could start only with a draft document and some early versions of the comments. Thu user in charge can see what he will have to do and start processing some comments. Note that in this case some synchronization will have to be done with the final version of the document.

These three strategies have an impact on the general flexibility of the process execution. While the first one is very open but can lead to a lot of inconsistencies, the last one is more rigid and remove most of the interest of anticipation. In the remainder of the paper, we will consider that the implemented policy is the second one.

Transition from *ready to anticipate* to *anticipating*. As soon as an activity becomes *ready to anticipate,* it is scheduled, i.e. it is assigned to all agents who actually qualify under its associated staff query. It is important to note that users know the state of the activity and can decide to anticipate. When one of these agents starts to anticipate, the activity passes in *anticipating* state and disappears from the *to do list* of the other agents.

Transition from *anticipating* to *executing*. An activity in *anticipating* state passes to *executing* state when it is in a situation where it would be allowed to start its execution if it was not anticipating; in other words, when previous activities have finished. Especially, it means that for all intermediate results it is anticipating with, it has got the corresponding final result. The user executing the task will have to do the corresponding update.

Transition from *ready to anticipate* to *executable*. An activity in *ready to anticipate* state passes to *executable* state in the same conditions as traditionally an activity passes from *initial* to *executable*. This situation corresponds to an activity that did not anticipate.

Transition from *anticipating* to *dead*. An activity in *anticipating* state passes to *dead* state if it is situated in a path that is not followed in the current workflow instance. Such a transition has the same motivations as for a traditional workflow activity to go from the *initial* state to the *dead* state. An anticipating activity makes the hypothesis that it will be executed. This is not sure. Thus, it is possible that the work executed by it become useless. However, we think that, due to the nature of the applications we consider, this situation will not occur frequently: the objective of anticipation is to make the right decision at the right time thanks to rapid feedback.

We can see from this description, that modifications of the workflow execution engine to provide anticipation are not very important. Anticipation impacts the execution model of an activity in the following way. It is necessary:

- to introduce two new states in the activity model: *ready_to_anticipate* and *anticipating states*. Consequently, it is necessary to assign activities in *ready_to_anticipate* state to agents who are able to start their anticipated execution,
- to modify the navigation algorithm: while in traditional workflow, the only event that triggers a new step for electing executable activities is the *termination of an activity*, in the modified model, *activity start* and *data production* events can change the state of succeeding activities to the *ready to anticipate* state,
- to modify data flow to integrate data exchange between anticipated activities (to manage exchange of intermediate results). In order to gain all the benefits of anticipation of activity execution, it is necessary to allow also early circulation of data between activities, i.e. publication of early or intermediate results in the output container of executing activities.

The last point (modification of data flow) is detailed in the next section

4.4 Data flow supporting intermediate results

As we consider mainly interactive activities, user can decide to provide output data before their end and possibly with several successive versions. New activity operations are introduced, *Write* and *Read*. These operations can be used by users (or even special tools) to manage publication of data during activity execution. *Write* operation updates an output element and makes it available to succeeding activities.

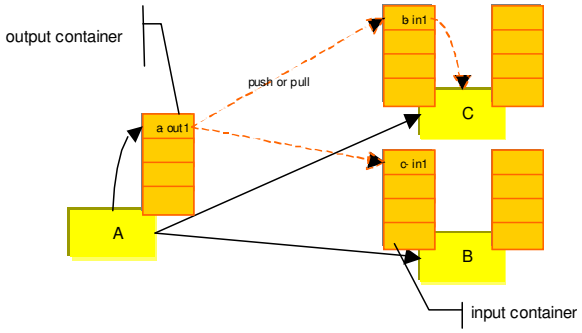

Figure 5 Data flow between an executing activity and anticipating activities

Consider the example of Figure 5 where activity A invokes the operation *Write(aout1)* to publish its output data *aout1*. Suppose that in data flow definition, two edges exist that have *aout1* as origin: an edge (aout1, bin1) between A and B activity and another one, (aout1, cin1) between A and C activity. If B or C is in anticipating state, they must be notified about the existence of new data (pull mode) or notified of arrival of data in their input container (push mode).

Read operation is used by anticipating activities in pull mode to update an input data with the new version published by the preceding activity. For unstructured data, a mechanism must be provided to synchronize with new versions (merge for instance). For text files, this is a common feature supported by version management systems.

Activities are no more isolated in black box transactions. They can provide results that can be used by succeeding activities. If the succeeding activities are interactive activities, the users in charge can consider taking this new value into account. They may also choose to wait for a more stable value supposed to arrive at a later time. Breaking activity isolation is necessary in order to benefit from the ability to anticipate but it may also cause some problems of inconsistency. These problems and the way to address them will be described in section 4.8.

4.5 Formal definition of anticipation

In this section we define the conditions for an activity to enter the "ready to anticipate" state for the control flow and data flow dictated strategies presented above. This specification extends the specification introduced in [20] for a traditional worflow. For a complete specification of COO-flow, see [14].

First, we need to introduce the definitions for *Activity state map* and *Data state map*[20], the applications that give the state of an activity (resp. a data element) at any moment during the execution of a process instance.

Definition 1 (***Activity state map***)

Let N be the set of all activities of a process model and $\aleph$ the set of natural numbers representing the time axis for each process instance ($0 \in \aleph$ represents the point in time when the associated instance of the process model is created). The activity state map

$\omega : \aleph \times N \rightarrow S$

associates at any point in time $i \in \aleph$ each activity A with its actual state $\omega(i, A)$.

S includes all activity states relevant to execution: *initial, executable, active, ready to anticipate, anticipating, terminated, completed, suspended.*

Definition 2 (***Data state map***)

Let V be the set of all process data elements (data needed as input by the activities, data required by conditions and the data to be exchanged between activities). The data state map

$\delta : \aleph \times V \rightarrow S$

associates at any point in time $i \in \aleph$ each data element v with its actual state δ (i, v).

The relevant states of a data element are *initial, intermediate and final*. An output data element of an activity enters the *intermediate* state when the activity executes a *Write* operation on this data; it enters the *final* state when activity completes.

Now we can define the conditions for an activity to enter the "ready to anticipate" state for the control flow and data flow dictated strategies presented in the previous section (for the free anticipation strategy, *initial* and *ready to anticipate* state are merged).

Definition 3 (***Activities in State "Ready to Anticipate", control flow strategy***)

Activity A becomes *ready to anticipate* at time $i \Leftrightarrow$

1. $\omega(i-1, A) = initial$
2. $\forall X \in A^{\leftarrow}, \omega(i, X) \in \{executing, anticipating\}$ if A is an AND-join node or a regular node

   $\exists X \in A^{\leftarrow}, \omega(i, X) \in \{executing, anticipating\}$ if A is an OR-join node

where $A^{\leftarrow}$ denotes the set of all direct predecessors of activity A.

A regular or an AND-Join activity in initial state enters the *ready to anticipate* state when its predecessors are

activated for execution or anticipation (*executing* or *anticipating* state). An OR-Join activity enters the *ready to anticipate* state when one of its predecessors is activated for execution or anticipation (active or anticipating state).

Definition 4 (***Activities in State "Ready to Anticipate", control and data flow strategy without conditions on data***)

The control and data flow strategy imposes supplementary conditions concerning the availability of input data as follows. Activity A becomes *ready to anticipate* at time $i \Leftrightarrow$

1. condition 1 of Definition 3
2. condition 2 of Definition 3

3. $\forall v \in i(A) : (w, v) \in \Delta^{in}(A) \Rightarrow \delta(i, w) \neq initial$ and $\exists v \in i(A) : (w, v) \in \Delta^{in}(A) \Rightarrow \delta(i-1, w) = initial$

where i(A) is the input container of activity A (the set of all its input data elements),

(w, v) is a data connector connecting a data element *w* of the output container of an activity ($w \in o(X)$) with a data element *v* in the input of another activity ($v \in i(A)$),

$\Delta^{in}(A)$ denotes the set of all data connectors (v, w) having as destination an element in the input container of activity A ($v \in i(A)$)

The third condition requires for all input elements ($v \in i(A)$) being destination of a data connector, the source of the data connector to be in *intermediate* or *final* state.

4.6 Anticipation and rapid feedback

Besides supporting early start, anticipation can be used to provide rapid feedback to preceding activities. A communication channel can be created backward of the defined flow of activities. As anticipation allows successive activities to execute partly in parallel, it is natural to imagine that people may have some direct feedback (e.g. comments) to provide to user of preceding activities.

The example of Figure 1 is the most obvious one: the *Review* activity can provide early comments while the *Edit* activity is still running. In this case, the *Review* activity may last longer. It will have to double check for the comments that may have been used by the *Edit* activity and remove them from its list. The *Modify* activity is shorter: some comments have been processed during the preceding phase.

4.7 Anticipation to increase parallelism between a process and its subprocess

In a traditional workflow, an activity is a black box that produces output data at the end of its execution. In our approach, an activity may fill an output parameter in its output container as soon as it is produced. These partial results become available to succeeding activities. They can enter anticipating state and initiate actions based on these results. For instance, let consider the activity of sending a letter to a customer; the letter can be prepared in anticipating state with available data and really sent out in executing state. In this way, the process execution is accelerated.

Anticipation is especially useful when the activity is a sub-process. The output container of the activity is the output container of the process implementing it. Similar to an activity, a process can gradually fill data produced by its activities in its output container. These data become available for subsequent activities; otherwise they would have to wait the end of the sub-process. Anticipation allows increasing the parallelism between the main process and the sub-process implementing the activity.

To illustrate this, we can use the example presented in section 2 depicted in Figure 3. The output data of the sub-process are the warehouse where the product is

available and the delivery date. The data are filled in the output container of the sub-process as soon as its corresponding activities produce them; succeeding activities in the top-level process can enter anticipating state.

As soon as the product is in stock or the date when it will be received from manufacturer is known, these data are filled in the output container of the sub-process and made available to succeeding activities. The acknowledgement to the customer can be prepared before the termination of the subprocess. Similarly, the *Prepare Shipment* is initiated as soon as it is known from which warehouse the product will be delivered and at what date it will be available.

For this kind of process, control and data flow dictated anticipation can be applied efficiently. However, the *Shipping* activity can not be anticipated. This is a typical activity that can be executed only when all the preceding activities are terminated.

4.8 Synchronization and recovery

Publishing intermediate results is important to take advantage of anticipation. However, an activity that has published results during its execution may fail or die. The problem is *how to compensate* the visibility of such results (it can be related to *dirty read* problem in traditional concurrency theory). In order to assure this, the following rules are applied**:**

1. An activity that has read an intermediate result must read the corresponding final result if necessary (it is different from the intermediate result).

   Synchronization with previous activities: An activity can enter completed state only from executing state. So, an anticipating activity, at a given moment will enter executing state. At this moment, previous activities will be completed and their final results available. When the activity enters executing state, its input container will be actualized with the final versions of its input parameters.

   Synchronization in case of feedback: Consider the example presented in Figure 1. If state transition assures that the *Review* activity will read the last version of the *Edit* activity, it doesn't assure that the last result of the comment will be read by the first activity. Based on the principle that users are aware of risks introduced by intermediate results, we make the hypothesis that the management of intermediate results for preceding activities (feedback purpose) can be completely let under the responsibility of user. It does not need to be explicitly managed by the workflow system.

   However, if feedback is important for the process, explicit loops can be defined in the process model. In the workflow model we base our work on, the repetition of a group of activities is specified by defining them as a block activity; the exit condition of this block activity then represents the proper condition for when to leave the loop.

   In our example, *Edit* and R*eview* activities would form a block activity. If the last comments were not taken into account, the exit condition of the block activity

would not be fulfilled and the process will be re-instantiated; the *Edit* activity would finally take into account the last comments.

2. An activity cannot produce intermediate results if it is not currently "certain" that it will enter the normal executing state. Independently of the anticipation strategy, we adopt a conservative approach concerning the moment when an anticipating activity may publish data.

   For this purpose, only anticipating activities satisfying the following conditions can publish anticipated results:

   - if the activity is a regular[5] (sequence) node, the predecessor must be in executing state,
   - if the activity is an or-join activity, it must exist a predecessor in the executing state,
   - if the activity is an and-join activity, predecessor activities must be in executing or completed state.

   These conditions ensure that this activity has a good chance to terminate, i.e. it should enter the executing state and not pass in *dead* state (because of the control flow). Of course, this is an optimistic approach: nothing ensures that the preceding activities will not be canceled neither that a human agent will not kill it.

3. In case where an executing activity is canceled, anticipating activities that published data may need to be compensated. As only direct successors could publish data, the influence of canceling an activity is limited. After the state change, the status of anticipating activities is recomputed. They may go to dead state; the work done in anticipating state is lost. Otherwise, they remain anticipating. New updated values will be provided by the execution of their preceding activities. They will have to resynchronize with the next valid input.

   As we can see, anticipation does not have an important impact on the general problem of workflow recovery, as long as activities do not have side effects. The work done in anticipating state may be lost but this is the price of flexibility. In case of activities having side effects outside of the scope of the workflow system, we are still obliged to introduce a special case described in the next section.

4.9 Suitability and applicability of anticipation

Anticipation is not suitable in any situation and not applicable for any activity type. For instance, it cannot be applied for an automatic activity having effects that cannot be compensated. However, it can be applied for automatic activities that are of retry type (flexible transactions [29]). For example, an activity that searches available flight tickets in a database, can be automatically restarted at each modification of its inputs if the execution speed of the process is more important for the application than

[5] regular activity: activity having one incoming edge

the overload created in the external database. A similar example is an activity that compiles a program at each modification of one of its modules.

During the definition phase of the process, it may not be possible to know in advance when anticipation will occur but it is possible to know which activities must not anticipate. These activities will be marked as such and will follow a more classical execution model.

Regarding the general strategy for the process execution, it must be decided when the process is instantiated and can be changed from one instance to the other. Depending on the level of flexibility required, free anticipation, control flow dictated anticipation or data flow dictated anticipation will be used during all the process. Usability studies are required here to know precisely the actual impact of each strategy.

In this part we have described how a simple modification of the classical model can enhance the flexibility of workflow execution in different ways. Now we are going to present how it is integrated in a larger framework to support cooperative workflow execution.

5 Implementation

We implemented a workflow execution engine allowing anticipation as part of the MOTU prototype whose goal is to provide a framework to support cooperative work of virtual teams. The prototype is written in Java[6].

The basis for the implementation is a non-linear version management server that provides the functionalities for public and private workspace management. This system has been extended with a basic workflow execution engine that implements anticipation (see Figure 6). Another component of the system is a cooperative transaction manager that allows exchanges of data between concurrently executing activities. Users are notified about the actions of other group members in their related projects: we consider that process and data awareness are important features for a cooperative environment.

Anticipation is implemented as part of the Workflow execution engine. Activities that are not anticipatable can be specified. A Coo Transaction is associated to each activity[13,10]. This Coo transaction provides support for optimistic concurrency control between activities. A Coo Transaction[2] has a private base (a workspace) that contains a copy of all the objects accessed (read and updated) by activities; it also serves as communication channel between successive activities.

[6] motu.sourceforge.net

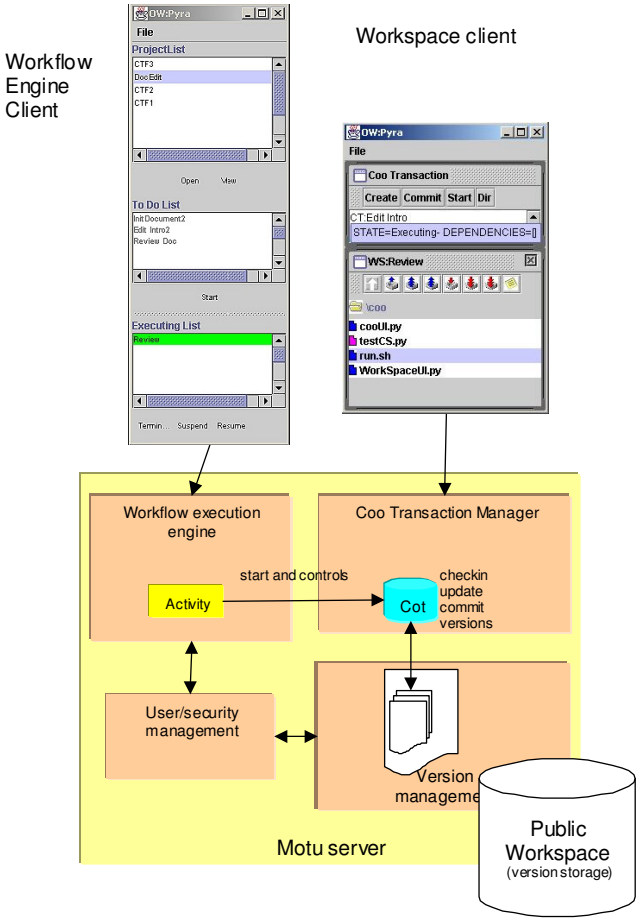

Figure 6 MOTU Basic architecture

Figure 7 shows the actual interface for three different users executing the process we used as example. The first one (V4) is actually executing the *Edit* operation. The second one (Pyra) is anticipating and has started the *Review* activity. *Review* appears in his executing list (list of activities that user started to execute) with a different color. Data exchanges are done through their private workspace. V4 can provide Pyra with drafts of its document. Pyra can transmit comments (feedback) using the same channel. In this case, Vortex could start to modify the document. *Modify* is in its To Do list as an anticipatable activity. The anticipation strategy for the process is the control flow dictated anticipation.

We have implemented a new version of the workflow model using *Enterprise Java Beans* and the *J2EE* (*Java™ 2 Platform, Enterprise Edition*), taking advantage of their features for security, transaction management, connection with databases systems and portability. This implementation uses the Jboss application server

(http://www.jboss.org/). It is called Bonita and can be downloaded from (http://freshmeat.net/projects/bonita-workflow).

Also, Bonita is being tailored for different purposes in LibreSource (http://www.libresource.org), a project dedicated to distributed software development, and in Coopera (http://woinville.loria.fr/coopera), a project dedicated to e-learning for children. COO-flow prototype is being integrated also in ToxicFarm (http://woinville.loria.fr), a platform hosting services for distributed teams. ToxicFarm will be experimented in the context of cooperative learning and of software design and development at the scale of France

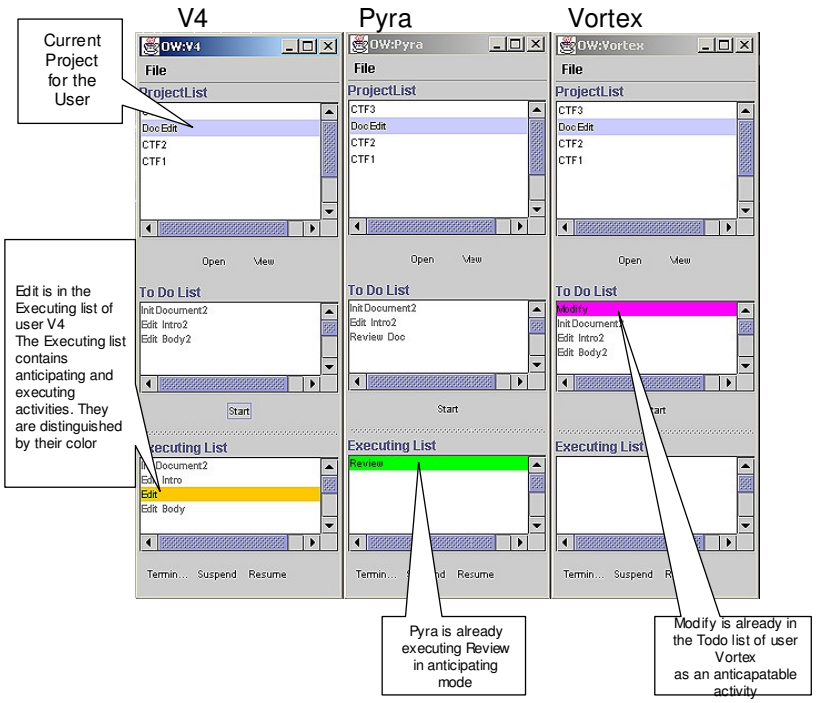

Figure 7 Actual execution of the example process

6 Conclusion

In this paper, we have presented a simple yet powerful way to modify workflow systems classical behavior to provide more flexible execution of processes. Compared to other approaches that try to provide flexibility in workflow models, the one we propose is simpler to understand for end users. This simplicity is essential in

cooperative processes allowing graphical representation, helping their participants to situate in the context of the process and facilitating manual interventions for dynamical modifications. We showed that extending existing systems with anticipation is simple since it requires just modifying the state transition management of the execution engine and some adaptation to the way data are transmitted between activities. Anticipation also provides substantial benefits regarding activities execution, especially for interactive activities:

- Parallelism of execution between successive activities
- Possibility of early feedback between successive activities
- Potential acceleration of the overall execution.

Of course, there is also the risk of doing some extra work but as we mainly target interactive activities, we believe that users are able to evaluate the opportunity to use anticipation and what they may gain from it. Furthermore, we plan to adapt the algorithms developed in [11] to use analysis of different executions of the same process with and without anticipation in order to help users to decide if anticipation should be used or not. This can be helpful in production processes where some paths in the workflow are taken most of the time given some initial conditions.

The next step of this work will be to consolidate the integration of the workflow model with cooperative transactions that we presented in [10]. Having as objective the simplicity of the workflow model, we will render the transactional support transparent to user, by embedding cooperative transactions in a set of cooperation patterns, at the same level than MCI patterns [24].

In the continuation of this study, an ongoing objective is to make usage analysis in realistic situations taking advantage of the applicative projects introduced above.

We are also concerned with the integration of the COO-flow model with other workflow models, in the context of an intranet process or a multi-enterprises process.

References

1. Agostini, A. and G. De Michelis, *Modeling the Document Flow within a Cooperative Process as a Resource for Action*, Technical Report CTL-DSI, 1996, University of Milano.
2. Canals, G., *et al.*, *COO Approach to Support Cooperation in Software Developments*. IEE Proceedings Software Engineering, 1998. **145**(2-3): p. 79-84.
3. Casati, F., *et al. Workflow Evolution*. in *15th Int. Conf. On Conceptual Modeling (ER'96)*. 1996.
4. Dayal, U., Hsu, M. and Ladin, R.: *Business Process Coordination: State of the Art, Trends, and Open Issue,. 27th International Conference on Very Large Data Bases (VLDB)*, 2001, Roma.
5. Ellis, C. and C. Maltzahn. *Chautaqua Workflow System*. in *30th Hawaii Int Conf. On System Sciences, Information System Track,*. 1997.
6. Fent, A., Reiter, H. and Freitag, H., *Design for Change: Evolving Workflow Specifications in ULTRAflow*, *CAISE 2002*, LNCS 2348, 516-534, 2002
7 Georgakopoulos, D. *Collaboration Process Management for Advanced Applications*. in *International Process Technology Workshop*. 1999.

8 Godart, C., O. Perrin, and H. Skaf. *coo: a Workflow Operator to Improve Cooperation Modeling in Virtual Processes*. in *9th Int. Workshop on Research Issues in Data Engineering Information technology for Virtual Entreprises (RIDEVE'99)*. 1999.

9 Grégori, N., *Etude clinique d'une situation de conception de produit. Vers une pragmatique de la conception.*, Ph.D. Thesis in Social Psychology , 1999, University of Nancy 2: Nancy. p. 239

10 Grigori, D., F. Charoy, and C. Godart. *COO-flow: a Process Technology to Support Cooperative Processes, International Journal of* of *Software Engineering and Knowledge Engineering (IJSEKE), Special Issue: Best Papers from SEKE 2003, Vol. 14, No. 1,January 2004*

11 Grigori, D., Casati, F., Dayal, U., and Shan, M.-C., *"Improving Business Process Quality through Exception Understanding, Prediction, and Preventing", VLDB'2001, 27th International Conference on Very Large Data Bases*, September 2001, Rome, 159-168

12 Grigori, D., Charoy, F. and Godart, C., *"Anticipation to Enhance Flexibility of Workflow Execution", DEXA'2001, 12th International Conference on Database and Expert Systems Applications, September 2001, Munich, LNCS 2113, 264-273*

13 Grigori, D., Charoy, F. and Godart, C., *Flexible Data Management and Execution to Support Cooperative Workflow: the COO approach*. in *The Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'01)*. 2001. Beijing, China

14 D. Grigori, Workflow elements for coperative processes definiton and enactment, PHD thesis in Computer Sciences. 2001, University Henri Poincaré Nancy1, Nancy. p. 135.

15 Hagen, C. and G. Alonso. Beyond the Black Box: Event-based Inter-Process Communication in Process Support Systems. in 9th International Conference on Distributed Computing Systems (ICDCS 99). 1999. Austin, Texas, USA.

16 Jablonski, S. *Mobile: A Modular Workflow Model and Architecture*. in *4th international Working Conference on Dynamic Modeling and Information Systems*. 1994. Noordwijkerhout, NL.

17 Jablonski, S. and C. Bussler, *Workflow management - Modeling Concepts, Architecture and implementation*. 1996: International Thomson Computer Press.

18 Joeris, G. *Defining Flexible Workflow Execution Behaviors*. in *Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications', GI Workshop Proceedings - Informatik'99, Ulmer Informatik Berichte Nr. 99-07, University of Ulm*. 1999.

19 M. Klein, C. Dellarocas and A. Bernstein (Eds.), Computer Supported Cooperative Work (CSCW) The Journal of Collaborative Computing, Vol. 9, Kluwer, 2000, special issue on Adaptive Workflow Systems.

20 Leymann, F. and D. Roller, *Production Workflow*. 1999: Prentice Hall.

21 Nutt, G.J. *The Evolution Toward Flexible Workflow Systems*. in *Distributed Systems Engineering*. 1996.

22 Reichert, M. and P. Dadam, *ADEPTflex - Supporting dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems*, 1998. **10**.

23 Rinderle, S. , Reichert, M. and Dadam, P., *Evaluation of Correctness Criteria for Dynamic Workflow Changes, Proc. Int'l Conf. on Business Process Management* (BPM '03), Eindhoven, Netherlands, June 2003

24 Schuster, H., Baker, D., Cichocki, *A.,* Georgakopoulos, *D. and* Rusinkiewicz, M., *The Collaboration Management Infrastructure, 16th International Conference on Data Engineering*, February 28 - March 03, 2000, San Diego, California

25 Suchmann, L.A., *Plans and Situated Action. The Problem of Human-Machine Communication*, in *Cambridge University Press*. 1987.
26 van der Aalst, Exterminating the dynamic change bug. A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297-317, 2001
27 Weske, M. *Flexible Modeling and Execution of Workflow Activities. in 31st Hawaii International Conference on System Sciences, Software Technology Track* (Vol VII). 1996.
28 Workflow Management Coalition, *The Workflow Reference Model*. 1995.
29 A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. *Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In Proc. ACM-SIGMOD International Conference on Management of Data*, 1994